

**Lab 01****Getting to Know Your Robot – Let's Get Moving!**

Read this entire lab procedure before starting the lab.

Purpose: The purpose of this lab is to confirm that you have all of the necessary software installed on your computer to program the robot. The secondary goal is to get the robot moving and to examine problems with raw odometry for pose estimation.

Objectives: At the conclusion of this lab, the student should be able to:

- Describe the primary components of the robot including the actuators, effectors, and sensors.
- Program the robot to move to a given angle or goal position
- Program the robot to move in a prescribed motion (circle, square, figure eight)
- Write properly commented, modular code
- Write a technical memo/worksheet to report the results of an experiment

Equipment: Base Robot

Masking Tape

Ruler or Tape Measure

3 LEDs and 3 220Ω resistors

Various Wires

Software: Arduino IDE: <https://www.arduino.cc/en/Main/Software>

PlatformIO IDE: <https://docs.platformio.org/en/stable/integration/ide/vscode.html>

References: Arduino Reference, Getting Started

<https://www.arduino.cc/en/Guide/HomePage>

AccelStepper library

<http://www.airspayce.com/mikem/arduino/AccelStepper/index.html>

How to install libraries in Arduino software

<https://www.arduino.cc/en/Guide/Libraries>

Arduino GIGA R1 Wifi

<https://docs.arduino.cc/hardware/giga-r1-wifi>

A4988 Step Stick Driver

<http://howtomechatronics.com/tutorials/arduino/how-to-control-stepper-motor-with-a4988-driver-and-arduino/>

**Pre-Lab:**

- Read this entire lab procedure before the lab recitation.
- Review Lab recitation slides on Moodle
- Review Lab recitation video on [ECE425 YouTube Playlist](#)
- Read Olson paper, [A Primer on Odometry and Motor Control](#).
- Review [AccelStepper Library](#) (see link in references)
- Write pseudocode for the functions: Go to Angle, Go to Goal, Circle, Figure Eight, Square. Note that Go To Goal and Go To Angle must use encoder feedback. Square must use Go To Goal and Go to Angle
- Upload software design plan to Gradescope

Theory:

Locomotion refers to moving a robot from place to place. *Odometry* is a means of implementing dead reckoning to determine a robot's position based upon the robot's prior position and the current heading and velocity. The advantages of this method are that it is self-contained, it always provides an estimate of position, and positions can be found anywhere along curved paths. The disadvantages are that the position error grows and require accurate measurement of wheel velocities over time.

There are several types of odometry error including systematic from unequal wheel diameters, misalignment of wheels, finite encoder resolution and finite encoder sampling rate. There is also non-systematic odometry error such as travel over uneven floors, unexpected objects on the floor and wheel slippage. Lastly, there are odometry errors such as imprecise measurements, inaccurate control models and immeasurable physical characteristics. These inaccuracies in dead reckoning cause a robot traversing a square path to yield a result similar to Figure 1. Thus, dead reckoning is most appropriate for short distances because of the error accumulation.

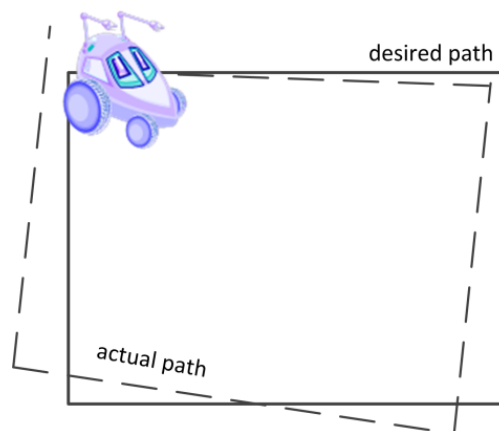


Figure 1: Robot Odometry Error



Most mobile robots contain wheels with an actuator. The actuator is typically a motor with an encoder to feedback information about how much and in what direction the motor shaft was rotated. This allows the robot to estimate its position relative to a start point. The GigaBot actually has stepper motors and they operate a little differently. Stepper motors operate by successively energizing the various coils of a motor in order to rotate a magnetic field through its different windings. This results in a shaft that rotates in discrete angular movements, called steps. In a stepper motor, a 360-degree wheel revolution is divided into a whole number of steps. Each step moves the motor some percentage of a revolution, either in a forward or reverse direction based on the stepping order. The GigaBot platform uses stepper motors that are designed with 200 steps per wheel revolution. That means each wheel in the GigaBot can step forwards or back with a minimum granularity of $360/200 = 1.8$ degrees per step. This is a rather 'fine' granularity from a locomotive standpoint. Even with 'locomotive precision' made possible by the fine stepping distance of the stepper motors, the fact remains that mechanical systems are not perfect. Wheels slip from the surface, and stepper motors might miss steps along the way. In addition, the topology of the terrain may exacerbate this problem. As a result, odometry measurements will accumulate errors over time.

For accuracy, an encoder or some other feedback sensor should be used to periodically null out or reset the accumulation error. Note that when you have a robot with no sensor feedback (encoders), dead reckoning is the only method for estimating the robot's position. Without the encoders there is no way to correct the position. In proportional control, use the encoder to correct odometry error such as when the wheels slip or overshoot a goal position.

An inertia measurement unit (IMU) is used to get the position of a robot in three-dimensional space. The MPU6050 IMU includes an accelerometer and gyroscope. It is a six-axis IMU because it has six degrees of freedom. The output is three accelerations from the accelerometer and three angular velocities from the gyroscope. The chip uses I2C (inter-integrated circuit) protocol for communication.

In this lab, you will use odometry concepts to implement the go-to-angle and go-to-goal behaviors on the robot. Remember you will never be able to correct for all odometry error and must learn to implement the most ideal solution considering this variable. In order to use feedback control to implement these behaviors it is necessary to have sensors to measure the robot's actual position to compare to the desired position in order to calculate the error input for the controller. Figure 2 demonstrates the necessary feedback control system.

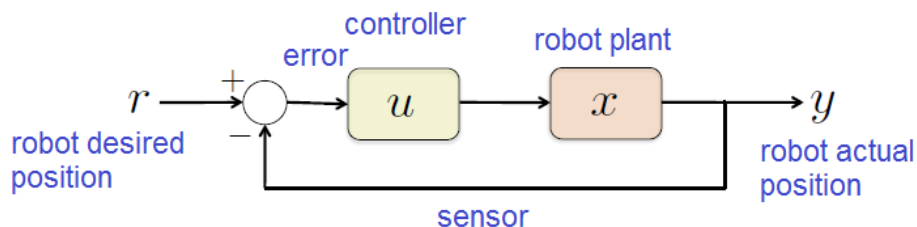


Figure 2: Robot Motion Feedback Control System



You can estimate the measured angle or goal by using velocity and time or number of steps to estimate position using forward kinematics. Figure 2 shows the dynamic model for implementing the Go-To-Goal behavior.

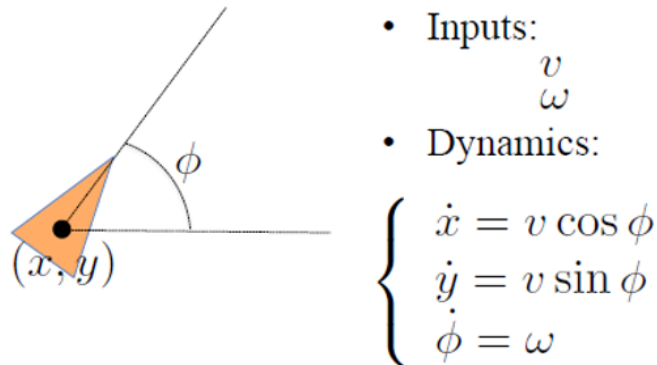


Figure 3: Unicycle robot dynamic model

In order to implement the Go-To-Goal behavior, the robot would calculate the angle to the goal point and use the Go-To-Angle behavior to turn first and then move forward to the goal. Therefore, the Go-To-Angle behavior should be implemented first. To calculate the desired turn angle, use the following formula.

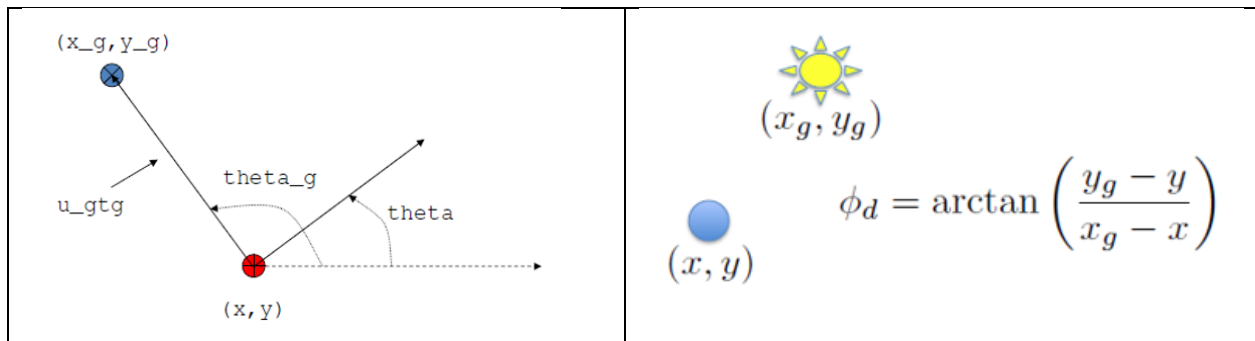


Figure 4: Go-To-Goal Notation

An alternative for implementing the Go-To-Goal behavior which may be a bit more difficult is to move the robot in a linear and angular motion at the same time as it converges on the goal point. In order to use this model, you will use the differential drive robot model to calculate the required motion given a constant linear velocity and variable angular velocity. This method would require the robot to incrementally calculate the vector to the goal position as it moves to converge on the position within some error. The Go-To-Goal behavior could be implemented by using a P controller to create the behavior, $\omega = K(\phi_d - \phi)$.



LAB PROCEDURE

Inventory

Your first task is to take an inventory of everything in your robot kit. The robot kit is shown in Figure 5, you must pack it back exactly like this before returning to the technician. You are responsible for returning the robot and all peripherals in the exact same condition that you received them in so if anything is missing, please notify your instructor or the ECE technician immediately. **Remember you are responsible to pay to replace any parts of the kit that are lost or damaged! You will receive an incomplete in this course and a hold on your business office account until this is done.**

- Robot with
 - Arduino Giga R1 Wifi
 - 2 sonar
 - 4 Infrared sensors
 - Left and right encoders on motor
 - HuskyLens Camera
- Wall Adapter Power Supply
- USB Cable
- Energizer Battery Charger
- 2 Rechargeable Batteries
- Pocket Screwdriver
- Flashlight
- Tape Measure
- 2 Photoresistors and 10k resistors
- 3 LEDs and 330Ω resistors
- MPU6050 Module 3 Axis Analog Gyro Sensors+ 3 Axis Accelerometer Module

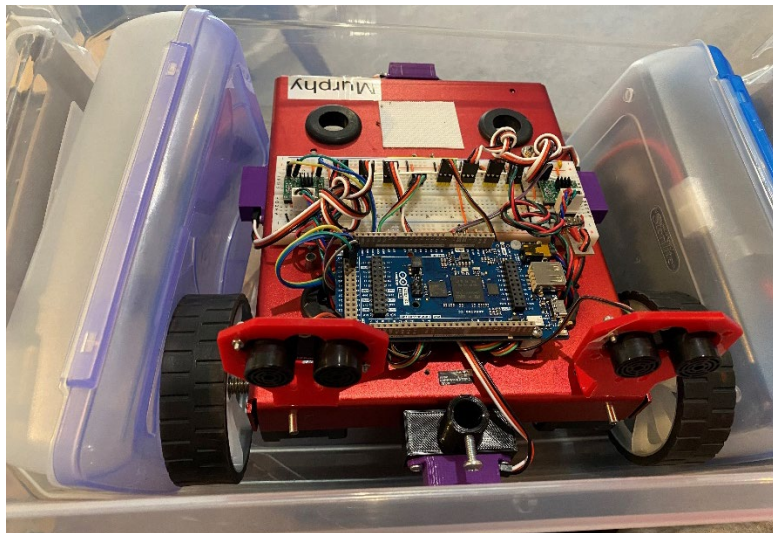


Figure 5: Robot Kit

Figure 6 highlights some of the key elements on the robot including the stepper motor driver, Arduino GIGA microcontroller board, IR sensors, sonar sensors, breadboard, motor power, and sensor power busses on the breadboard. You will add more peripherals as the quarter progresses.

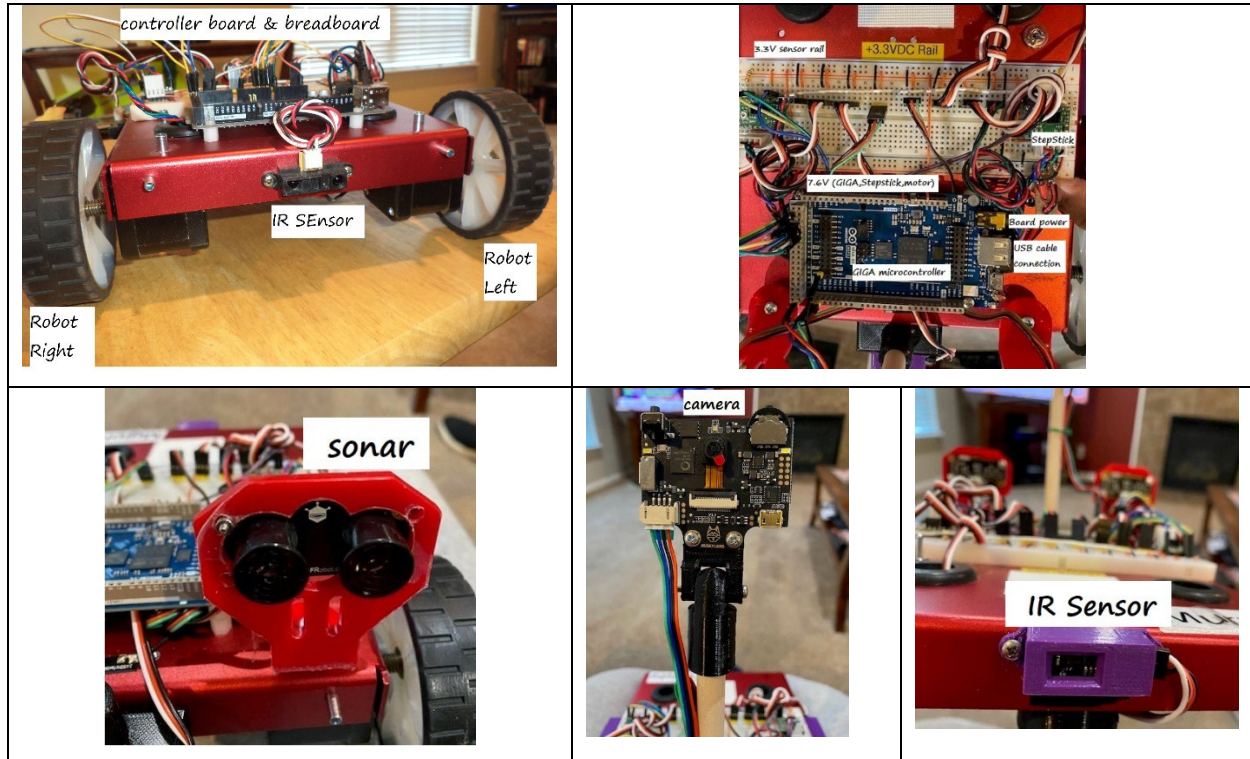


Figure 6: Robot Hardware

Part 1 – Software Installation

In this section you will install the software to program your robot. You should use Arduino or PlatformIO IDE to program the robot. The Arduino IDE method is described below but you will have to do independent research to use PlatformIO.

Arduino IDE

1. Navigate to the Arduino website, <https://www.arduino.cc/>, and click on Download.
2. Download the latest version of the Arduino IDE to your computer.
3. Go to the Moodle course website and download the Lab 01 files. The files are an Arduino sketch to help get the robot moving and the *AccelStepper* library.
4. Install the following libraries in the Arduino IDE by following the directions [here](#). Install the following libraries in the PlatformIO IDE by following the directions [here](#).
 - Arduino
 - AccelStepper
 - MultiStepper
5. Review the documentation and examples in the *AccelStepper* library folder to understand what it does. It is most important to understand the functions that use



blocking or reset speed to zero. *Look at the Blocking, Bounce, ConstantSpeed, MultipleSteppers, MultiStepper, Overshoot, Random, ProportionalControl, Quickstop, and Random Examples.*

PlatformIO IDE

1. Navigate to the PlatformIO website, <https://platformio.org/install>, and click to download the PlatformIO IDE.
2. Download VSCode for Windows. Accept all default selections.
3. Open Visual Studio Code and click on the 4 squares in the bottom left corner which represents Extensions.
4. Type “platformio” in the search box and select the “PlatformIO IDE” from the results. Select the Install button and wait for the IDE to finish installing.
5. Close and re-open Visual Studio to confirm that all extensions are installed.
6. Click on the PlatformIO IDE icon and click create new project.
 - **Project Name** – RobotName_Lab01
 - **Board** – The type of microcontroller board you are using.
 - **Framework** – The framework must match the board, and PlatformIO will determine this automatically for you once you select a board.
 - **Location** – Where you want your files stored. You can just leave this checked to accept the default location or uncheck it to choose a specific one.
7. Since the GIGA is a new board it is not yet available on PlatformIO so you will need to research some work arounds if you want to go with this method.
8. Review the PlatformIO documentation for the proper way to set Project Settings.

Part 2 – Wire the Robot

1. Wire the robot motor wiring to match the pin numbers on the following table.

Table 1: Motor Wiring

	Left Motor	Right Motor
Stepper enable pin	48	48
Step pin	52	50
Direction pin	53	51

2. Put the LED in series with a 100, 220 or 330 ohm resistor and connect it to the pin shown in Table 2. The long leg of the LED goes to the pin, the short leg (or flat rim on LED cap) goes to the resistor and the other leg of the resistor goes to ground.
3. Wire the left encoder to pin 18 and the right encoder to pin 19 as shown in Table 3.



Table 2: LED Wiring

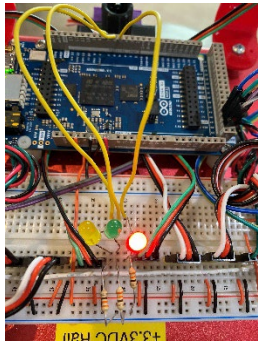
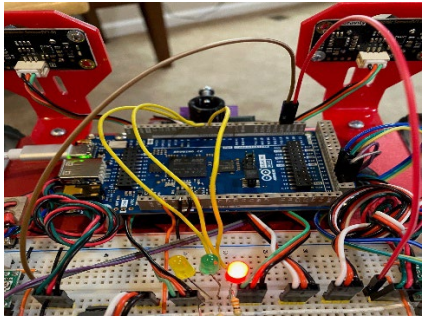
Red LED	Pin 5	
Green LED	Pin 6	
Yellow LED	Pin 7	

Table 3: Encoder Wiring

Left Encoder	Pin 18	
Right Encoder	Pin 19	

Part 3 – Robot Motion

In this section, you will run the program to move your robot.

1. Remember to always insert a 5 second delay at the beginning of all your code to give you time to unplug the USB cable and put the robot on the floor or get it on the test stand. You should also use LEDs to illuminate to indicate when the robot is about to move. You should also always use a test stand or the plastic box as a test stand to drive the robot when testing on the table.
2. The robot uses the A4988 Stepper Motor Driver StepStick to drive the stepper motors, view the following link for more details on the code.
<http://howtomechatronics.com/tutorials/arduino/how-to-control-stepper-motor-with-a4988-driver-and-arduino/>
3. Download “**RobotIntro.ino**” and read the comments at the top.
4. This file is the beginning sample code to help you get the robot moving. It also shows the proper way to comment your code. You will need to modify the comments, add additional comments, create new functions and add and modify the code based upon the lab requirements.



5. Make sure you read all the comments and understand what the code is doing. Make sure to ask your professor or the TA if you are not sure about any of it.
6. Rename and save this file as **"YourRobotName-Lab01.ino"**.
7. Use the USB cord to connect the robot microcontroller to your computer and wait for the driver to install.
8. You may have to install the FTDI driver if it is not installed automatically to recognize that your board is connected, <https://support.arduino.cc/hc/en-us/articles/4411305694610-Install-or-update-FTDI-drivers>
9. Use device manager to find the COM Port where the microcontroller was installed
10. In the Arduino IDE click Tools->Board->Arduino/Arduino GIGA R1
11. In the Arduino IDE, click Tools->Port->COM# (where board is connected, see Device Manager)
12. In Arduino IDE, click Sketch->Verify/Compile (CTRL-R) or click check icon
13. In Arduino IDE, click Sketch->Upload (CTRL-U) or click right arrow icon
14. Use the toggle switch to turn on power to the robot motors. The switch turns on power to the board and robot motors when the switch is back toward the label and tail wheel.
15. **WATCH THE ROBOT! DON'T LET IT DRIVE OFF THE TABLE! USE A TEST STAND! A BOOK OR BOX!**
16. If the motors don't move either the batteries need to be charged or the motors are wired incorrectly.
17. The robot should continue to move forward and backward one full rotation until you turn the power off.
18. Change the *stepTime* variable to a different value between 300 and 4000 and observe how this changes the robot behavior. **Comment on this observation in your lab memo.**
19. Now comment out the first function, *move1()*, in the *loop()* function and uncomment the next second function, *move2()*, in the main loop.
20. Compile and upload the program to see the difference in the code and robot behavior. **Comment on this observation in your lab memo.**
21. Next change the library functions to some of the other library function options listed in the comments and make observations about the change in robot behavior. The function options are *move()*, *moveTo()*, *stop()*, *run()*, *runSpeed()*, *runToPosition()*, *runToNewPosition()* and *runSpeedToPosition()*. **Comment on these observations in your lab memo.**
22. Now comment out the *move2()* function and uncomment the *move3()* function. Repeat steps 11 and 12. **Comment on your observations in your lab memo.**



23. Next repeat steps 11 and 12 for the *move4()*, *move5()*, and *move6()* functions.
24. Finally, this example code with header comment, block and in line comments and modularity with functions is how you should write your programs. It is very important to create functions and re-use code as much as possible for subsequent labs. In addition, if your code is not well commented it is impossible for you to understand later and for me to follow when grading it. Your code grade will be determined by how well your code is organized, commented and modularity.
25. Confirm the following LEDS turn on in Table 4 when each function is active. If they do not, you need to check your wiring.

Table 4: Move function LEDS

<i>move1()</i>	RED LED
<i>move2()</i>	GREEN LED
<i>move3()</i>	YELLOW LED
<i>move4()</i>	RED and GREEN LEDs
<i>move5()</i>	GREEN and YELLOW LEDs
<i>move6()</i>	RED, GREEN, YELLOW LEDs

Part 4 – AccelStepper Library

In this section, you will examine the different functions in the AccelStepper library that you can use to move your robot.

1. Run each of the example programs from the AccelStepper library to understand what the various functions do. These examples are in the AccelStepper library zip folder.
2. You can access them from the IDE by click File->Examples->AccelStepper->
 - Blocking
 - Bounce
 - ConstantSpeed
 - MultipleSteppers
 - MultiStepper
 - Overshoot
 - ProportionalControl
 - Quickstop
 - Random
3. Comment in your lab memo on what you think each of the functions do.



Part 5 – Encoders

In this section you will confirm that the encoders attached to the left and right wheel are working correctly. I have provided the interrupt for the encoders in the “**RobotIntro.ino**” file.

1. Your robot has been outfitted with encoders to enable feedback to correct for odometry error between the desired number of steps and actual number of steps. This can be used to move the robot to a given angle or position.
2. Upload the program to the robot and open the serial monitor to examine the encoder ticks as the robot moves. If the encoder values are not changing as the motors move then check your wiring.
3. In the main *loop()* function vary the move direction and amount to determine how the encoder values change. Record the average encoder ticks for a quarter, half, full, and two rotations. How consistent are the values between movements? How consistent are the values between the left and right motor? Do the numbers go positive and negative as the robot wheels turn forward and backward? **Comment on the results in your lab menu.**

Part 6 – Motion Functions – DEMO DUE NEXT CLASS

In this section you will create some frequently used robot motion functions. I have provided the template for these functions in the “**RobotIntro.ino**” file. You have the flexibility of modifying how the function works and what input it takes but the functions must be there in your final submission. You will turn on a different LED for each function in order to get in the habit of using LEDs to indicate the robot state.

1. A robot forward is when the wheels move in the same direction at same speed. Write a *forward()* function for the robot where the input is the distance or forward movement time or speed. You have the flexibility of how you want to write the function and can modify the template to meet your needs. Test your robot, how many steps does it take to move forward 2 feet? You can measure the diameter of your wheel as a starting point to estimate this value. **State this value in your lab memo.**
2. A robot reverse is when the wheels move in the same direction at same speed. Write a *reverse()* function for the robot where the input is the distance or reverse time or speed. You have the flexibility of how you want to write the function and can modify the template to meet your needs.
3. A robot stop is obviously when the robot stops moving. Write a *stop()* function.
4. A robot spin is when the wheels move in opposite directions at the same speed. Write a *spin()* function for the robot where the input is the direction of the spin or spin time or speed. You have the flexibility of how you want to write the function and can modify the template to meet your needs. Test your robot, how many steps does it take to spin the robot 90 degrees? **State this value in your lab memo.**



5. A robot pivot is when one wheel is stationary and the other wheel moves forward or backward. Write a *pivot()* function for the robot where the input is either the direction or which wheel should move forward or backward or pivot time or speed. You have the flexibility of how you want to write the function and can modify the template to meet your needs. Test your robot, how many steps does it take to pivot the robot 90 degrees? **State this value in your lab memo.**
6. A robot turn is when the wheels move in the same direction at different speeds. Write a *turn()* function for the robot where the input is the direction of the turn or turn time or velocity differential. You have the flexibility of how you want to write the function and can modify the template to meet your needs.
7. *Hint: Use runSpeed() or runSpeedToPosition() for the turn function. You will need to call them in a loop() to continuously increment the steps. You will also need to setSpeed() right before the run function. If setSpeed() does not seem to be working then use setMaxSpeed().*
8. Figure 7 gives an example of the robot motions. You must demonstrate Part 6 of the lab during the next class session.

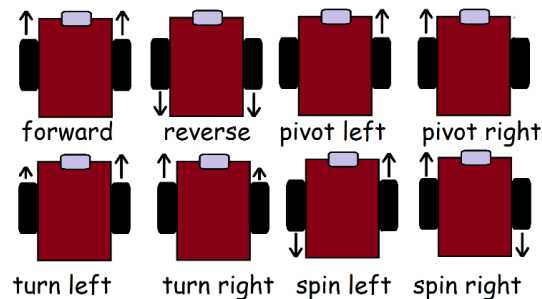


Figure 7: Robot Motion Summary

Part 7 – Circle and Figure 8 – DEMO DUE IN ONE WEEK

1. Write a “*moveCircle()*” function to move the robot in a circle with the diameter and direction as the input (see Figure 8). You should call the *moveCircle()* function from the *loop()* function. Turn on the RED LED when the robot is moving in a circle. **The circle diameter is measured from the center of the robot halfway between the wheels.**
2. Try to adjust the code so that the robot will start and end at the same point. **Comment on the results of this task in your memo.**

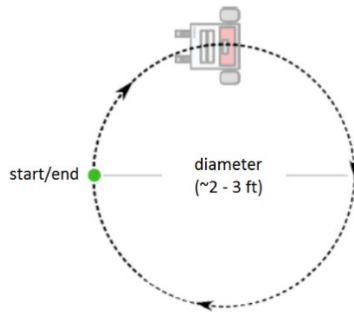


Figure 8: Circle Robot Motion

3. Next, create a `moveFigure8()` function to move the robot in a figure eight using two circles (see Figure 9).
4. Try to adjust the code so that the robot passes through the same center point of the figure 8 each time. You should call the `moveCircle()` function twice to create the `moveFigure8()` function. This means you may need to modify the `moveCircle()` function pass it a direction variable. You should call the `moveFigure8()` function from the `loop()` function. Turn on the RED LED and YELLOW LED when the robot is moving in a figure 8.

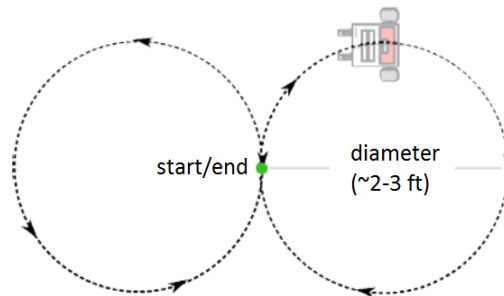


Figure 9: Figure Eight Robot Motion

Part 8 – Go To Angle Behavior - DEMO DUE IN ONE WEEK

1. In the program for part 5, there is a template for a go to angle behavior, create a method to implement the Go-To-Angle behavior. *Use the `atan2(y,x)` function to put your angles in the correct quadrant.*
2. The user will input a given angle in degrees and the robot should spin or pivot to the given angle. The robot should be able to take negative and positive angles to move clockwise or counterclockwise. You may have to add a variable to the function to indicate whether the robot should move left or right. Remember to use the right hand rule, positive angles move the robot left.
3. Take measurements to estimate the accuracy of the Go-To-Angle behavior. **Comment on the results in your lab memo.**
4. Your code should be modular with the `goToAngle()` written as a function that is called from the `loop()` function.



- The reference frame for the robot is shown in Figure 10, where the robot's start position is (x,y,θ) $(0,0,0)$. Note that the z axis is coming up out of the robot centered between the two wheels. The x axis is straight ahead and the y -axis is to the left. Positive angles are counterclockwise and negative angles are clockwise. Turn on the GREEN LED when the robot is executing the Go To Angle behavior.

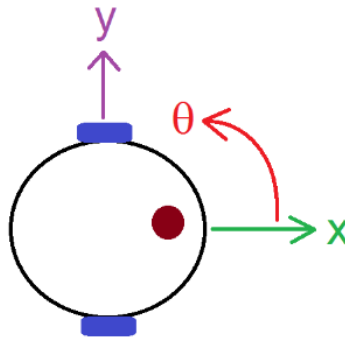


Figure 10: Robot Relative Reference Frame

Part 9 – Go To Goal Behavior DEMO DUE IN ONE WEEK

- Create a method to implement the Go-To-Goal behavior.
- The robot should move to the given goal position and stop within a certain error.
- Take measurements to estimate the accuracy of the Go-To-Goal behavior.
- Your code should be modular with the *goToGoal()* written as a function that is called from the *loop()* function when the button is pressed. The *goToGoal()* function could use the *goToAngle()* and *goToGoal* functions or some of the other primitive move functions.
- Use the reference frame shown in Figure 8 for the *goToGoal()* behavior.
- Turn on the GREEN LED and YELLOW LED when the robot executes the Go To Goal behavior.

Part 10 – Square Path - DEMO DUE IN ONE WEEK

Now that you have functions to create robot motion, you will create a function to move the robot in a square.

- Write a program to move the robot in a square path with sides specified in the function (see Figure 11).
- To correct for odometry error, compare encoder ticks and desired steps to stop the robot at the corners of the square. Alternately, you could use encoder ticks to move robot and command it when to stop. You can implement a proportional controller that compares the desired steps and encoder ticks in order to correct for odometry in the square function. ***Discuss this implementation in the lab worksheet.***



3. The way to implement this is to determine how many encoder ticks represent one foot forward on the left and right motor. If the command to move forward one foot is short or over by a few encoder ticks then move the robot forward and back to correct.
4. The *moveSquare()* function should be called from the *loop()* function and the input can be the length of the sides. Turn on the RED, GREEN, and YELLOW LED when the robot executes the Square behavior.

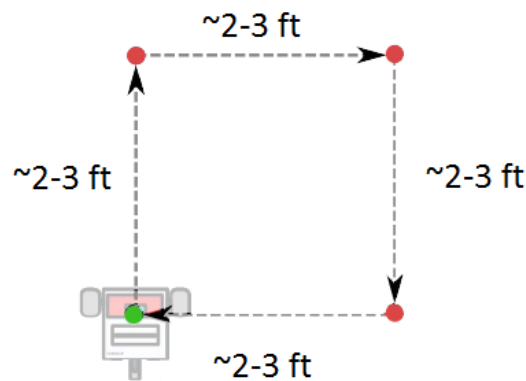


Figure 11: Square Robot Motion

Submission Requirements:

Software Design Plan

For each lab you will submit a software design plan which may be in the form of pseudocode, a flowchart, state machine or subsumption architecture. You will show this plan to the instructor during class, the plan will be graded and you will get feedback on implementation before designing the full system.

Demonstration:

Bring your robot fully charged to class every day! Plug it in overnight.

During the demonstration you will be graded on the following requirements:

- Ability to describe how the code works (commenting, organization, modularity)
- Robot move functions (forward, reverse, turn, pivot, spin, stop)
- Robot moves in a circle (RED LED)
- Robot moves in a figure eight (RED, YELLOW LED)
- Robot can move to angle given inputs such as -60° or 120° (GREEN LED)
- Robot can move to goal given inputs such as (12 inches, 6 inches) or (0 ft, 3 ft) (GREEN, YELLOW LED)



- Robot moves in a square with given side length (inches, feet) (RED, GREEN, YELLOW LED)

Code:

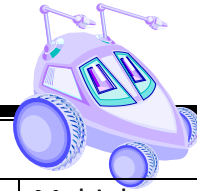
Use the following guidelines for creating your code.

- Code has a header comment with
- Program Name
 - Author Names
 - Date Created
 - Overall program description
 - Summary of key functions created with a description of each
- Code is modular with multiple functions called from the loop function. Functions have logical names that describe their functionality.
- Code is organized in a logical fashion with global and local variables with intuitive names.
- Code has block comments to introduce and describe each function and sufficient in line comments to describe how key parts of the code work.
- All functions and variables should have logical names that describe what they do or the value they hold. There should be no magic numbers or numeric values not related to a constant or variable to make it clear what it does.
- In general, someone unfamiliar with your project should be able to read your code and understand the functionality based upon your comments and naming convention.

Grading Rubric:

The lab is worth a total of 40 (prelab 5 pts, code, 10 pts, demo 10 pts, worksheet 15 pts). points and is graded by the following rubric. The grading rubrics is shown on the following table.

Points	Demonstration	Code	Memo
Excellent 100%	Excellent work, the robot performs exactly as required	Properly commented with a header and function comments, easy to follow with modular components	Follows all guidelines and answers all questions posed
Average 75%	Performs most of the functionality with minor failures	Partial comments and/or not modular with objects	Does not answer some questions and/or has spelling, grammatical, content errors



Poor 50%	Performs some of the functionality but with major failures or parts missing	No comments, not modular, not easy to follow	Multiple grammatical, format, content, spelling errors, questions not answered
Missing	Meets none of the design specifications or not submitted	Not submitted	Not submitted

You must submit you properly commented code in the appendix of the Lab Worksheet in text form. The Lab worksheet will be uploaded to GradeScope by **11:59 pm on Sunday after all demos are complete**. Your code should be modular with functions and classes in order to make it more readable.